Dataset Link:

https://www.kaggle.com/manishshah120/facial-expression-recog-image-ver-of-fercdataset

In [1]:
```
project_name = 'face-exp-resnet'
```

In [2]:
```
import os
import torch
import torchvision
import tarfile
import torch.nn as nn
import numpy as np
from PIL import Image
import torch.nn.functional as F
from torchvision.datasets import ImageFolder
import torchvision.models as models
from torch.utils.data import DataLoader
import torchvision.transforms as tt
from torch.utils.data import random_split
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
import torchvision.models as models
%matplotlib inline
```

In [3]:
```
data_dir = '../input/facial-expression-recog-image-ver-of-fercdataset/Dataset'
print(os.listdir(data_dir))
classes = os.listdir(data_dir + "/train")
print(classes)
len
```

```
['labels.txt', 'test', 'train']
['neutral', 'sadness', 'anger', 'disgust', 'fear', 'surprise', 'happiness']
```

In [ ]:
```
len(os.listdir(path+'/train/sadness'))
```

In [4]:
```
train_tfms = tt.Compose([
#                         tt.RandomCrop(32, padding=4, padding_mode='reflect'),
                        tt.RandomHorizontalFlip(),
                        tt.RandomRotation(30),
                        tt.ColorJitter(brightness=0.1, contrast=0.25, saturation=0.35
                        tt.RandomRotation(10, resample=False, expand=False, center=No
                        tt.ToTensor()
                       ])

valid_tfms = tt.Compose([tt.ToTensor()])
```

In [5]:
```
train_ds = ImageFolder(data_dir+'/train', train_tfms)
valid_ds = ImageFolder(data_dir+'/test', valid_tfms)
```

In [6]:
```
batch_size = 128
```

In [7]:
```
train_dl = DataLoader(
    train_ds,
    batch_size,
```

```
    shuffle=True,
    num_workers=4,
    pin_memory=True
                    )

valid_dl = DataLoader(
    valid_ds,
    batch_size*2,
    num_workers=4,
    pin_memory=True
                    )
```

In [8]:
```
def show_batch(dl):
    for images, labels in dl:
        fig, ax = plt.subplots(figsize=(16, 12))
        ax.set_xticks([])
        ax.set_yticks([])
        ax.imshow(make_grid(images[:64], nrow=16).permute(1, 2, 0))
        break
```

In [9]:
```
show_batch(train_dl)
```

In [10]:
```
def get_default_device():
    """Pick GPU if available, else CPU"""
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    """Move tensor(s) to chosen device"""
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    """Wrap a dataloader to move data to a device"""
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        """Yield a batch of data after moving it to device"""
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
```

```
        """Number of batches"""
        return len(self.dl)
```

In [11]:
```
device = get_default_device()
device
```

Out[11]:
```
device(type='cuda')
```

In [12]:
```
train_dl = DeviceDataLoader(train_dl, device)
valid_dl = DeviceDataLoader(valid_dl, device)
```

In [13]:
```
def accuracy(outputs, labels):
    _, preds = torch.max(outputs, dim=1)
    return torch.tensor(torch.sum(preds == labels).item() / len(preds))

class ImageClassification(nn.Module):
    def training_step(self, batch):
        images, labels = batch
        out = self(images)                  # Generate predictions
        loss = F.cross_entropy(out, labels) # Calculate loss
        return loss

    def validation_step(self, batch):
        images, labels = batch
        out = self(images)                    # Generate predictions
        loss = F.cross_entropy(out, labels)   # Calculate loss
        acc = accuracy(out, labels)           # Calculate accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()   # Combine losses
        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()      # Combine accuracies
        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_
            epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result
```

In [15]:
```
class ResNet18(ImageClassification):
    def __init__(self, num_classes):
        super().__init__()

        self.network = models.resnet18(pretrained=True)
        num_ftrs = self.network.fc.in_features
        self.network.fc = nn.Linear(num_ftrs, num_classes)

    def forward(self, x):
        return self.network(x)
```

In [16]:
```
model = ResNet18(7)
model = to_device(model, device)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-5c106cde.pth" to /root/.ca
che/torch/checkpoints/resnet18-5c106cde.pth
HBox(children=(FloatProgress(value=0.0, max=46827520.0), HTML(value='')))
```

In [17]:
```
pip install jovian --upgrade
```

```
Collecting jovian
  Downloading jovian-0.2.18-py2.py3-none-any.whl (64 kB)
     |████████████████████████████████| 64 kB 1.2 MB/s eta 0:00:011
Requirement already satisfied, skipping upgrade: pyyaml in /opt/conda/lib/python3.7/s
ite-packages (from jovian) (5.3.1)
Collecting uuid
  Downloading uuid-1.30.tar.gz (5.8 kB)
Requirement already satisfied, skipping upgrade: requests in /opt/conda/lib/python3.
7/site-packages (from jovian) (2.23.0)
Requirement already satisfied, skipping upgrade: click in /opt/conda/lib/python3.7/si
te-packages (from jovian) (7.1.1)
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in /opt/conda/lib/
python3.7/site-packages (from requests->jovian) (3.0.4)
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /opt/conda/lib/pytho
n3.7/site-packages (from requests->jovian) (2.9)
Requirement already satisfied, skipping upgrade: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.2
1.1 in /opt/conda/lib/python3.7/site-packages (from requests->jovian) (1.24.3)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /opt/conda/li
b/python3.7/site-packages (from requests->jovian) (2020.6.20)
Building wheels for collected packages: uuid
  Building wheel for uuid (setup.py) ... done
  Created wheel for uuid: filename=uuid-1.30-py3-none-any.whl size=6500 sha256=d80a66
6702e4abee7175db6b41cae4cad442545cb3da348942b89ba1db09dd0b
  Stored in directory: /root/.cache/pip/wheels/2a/ea/87/dd57f1ecb4f0752f3e1dbf958ebf8
b36d920d190425bcdc24d
Successfully built uuid
Installing collected packages: uuid, jovian
Successfully installed jovian-0.2.18 uuid-1.30
WARNING: You are using pip version 20.2.1; however, version 20.2.2 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgr
ade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

In [20]:
```
from tqdm.notebook import tqdm
```

In [21]:
```
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit_one_cycle(epochs, max_lr, model, train_loader, val_loader,
                  weight_decay=0, grad_clip=None, opt_func=torch.optim.Adam):
    torch.cuda.empty_cache()
    history = []

    # Set up cutom optimizer with weight decay
    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # Set up one-cycle learning rate scheduler
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs,
                                                steps_per_epoch=len(train_loader))

    for epoch in range(epochs):
```

```
        # Training Phase
        model.train()
        train_losses = []
        lrs = []
        for batch in tqdm(train_loader):
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # Gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # Record & update learning rate
            lrs.append(get_lr(optimizer))
            sched.step()

        # Validation phase
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

In [22]:
```
history = [evaluate(model, valid_dl)]
history
```

Out[22]:
```
[{'val_loss': 2.2266204357147217, 'val_acc': 0.10494791716337204}]
```

In [23]:
```
epochs = 25
max_lr = 0.05
grad_clip = 0.1
weight_decay = 1e-4
opt_func = torch.optim.Adam
```

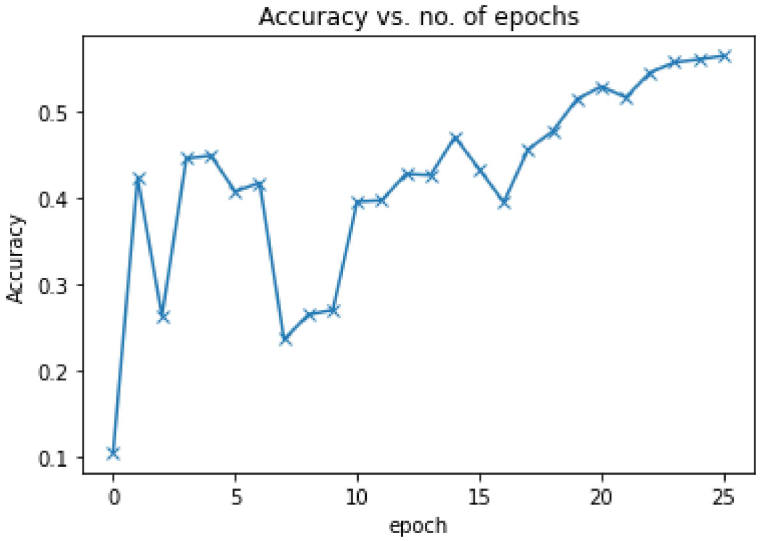
In [24]:
```
%%time
history += fit_one_cycle(epochs, max_lr, model, train_dl, valid_dl, grad_clip=grad_cli
```

```
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [0], last_lr: 0.00406, train_loss: 1.7168, val_loss: 4.2589, val_acc: 0.4231
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [1], last_lr: 0.00992, train_loss: 1.5858, val_loss: 1.8258, val_acc: 0.2629
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [2], last_lr: 0.01856, train_loss: 1.5872, val_loss: 1.4477, val_acc: 0.4449
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [3], last_lr: 0.02849, train_loss: 1.4801, val_loss: 1.4919, val_acc: 0.4484
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [4], last_lr: 0.03799, train_loss: 1.4782, val_loss: 1.5611, val_acc: 0.4067
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [5], last_lr: 0.04541, train_loss: 1.4874, val_loss: 1.5304, val_acc: 0.4169
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [6], last_lr: 0.04947, train_loss: 1.5076, val_loss: 2.4150, val_acc: 0.2359
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [7], last_lr: 0.04990, train_loss: 1.5150, val_loss: 1.7678, val_acc: 0.2645
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
```

```
Epoch [8], last_lr: 0.04910, train_loss: 1.5049, val_loss: 1.9986, val_acc: 0.2697
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [9], last_lr: 0.04752, train_loss: 1.4977, val_loss: 1.5829, val_acc: 0.3949
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [10], last_lr: 0.04523, train_loss: 1.4940, val_loss: 1.5521, val_acc: 0.3965
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [11], last_lr: 0.04228, train_loss: 1.4849, val_loss: 1.5122, val_acc: 0.4272
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [12], last_lr: 0.03877, train_loss: 1.4740, val_loss: 1.5201, val_acc: 0.4256
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [13], last_lr: 0.03483, train_loss: 1.4623, val_loss: 1.3770, val_acc: 0.4697
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [14], last_lr: 0.03056, train_loss: 1.4475, val_loss: 1.5268, val_acc: 0.4324
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [15], last_lr: 0.02612, train_loss: 1.4284, val_loss: 1.5852, val_acc: 0.3944
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [16], last_lr: 0.02164, train_loss: 1.4154, val_loss: 1.5172, val_acc: 0.4556
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [17], last_lr: 0.01727, train_loss: 1.3973, val_loss: 1.3563, val_acc: 0.4762
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [18], last_lr: 0.01315, train_loss: 1.3686, val_loss: 1.2933, val_acc: 0.5135
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [19], last_lr: 0.00941, train_loss: 1.3465, val_loss: 1.2411, val_acc: 0.5283
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [20], last_lr: 0.00617, train_loss: 1.3254, val_loss: 1.2951, val_acc: 0.5156
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [21], last_lr: 0.00354, train_loss: 1.2956, val_loss: 1.2189, val_acc: 0.5445
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [22], last_lr: 0.00159, train_loss: 1.2740, val_loss: 1.1882, val_acc: 0.5562
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [23], last_lr: 0.00040, train_loss: 1.2579, val_loss: 1.1670, val_acc: 0.5596
HBox(children=(FloatProgress(value=0.0, max=253.0), HTML(value='')))
Epoch [24], last_lr: 0.00000, train_loss: 1.2494, val_loss: 1.1639, val_acc: 0.5638
CPU times: user 3min 19s, sys: 16.2 s, total: 3min 35s
Wall time: 13min 23s
```

In [25]:
```
def plot_acc(history):
    acc = [x['val_acc'] for x in history]
    plt.plot(acc,'-x')
    plt.xlabel('epoch')
    plt.ylabel('Accuracy')
    plt.title('Accuracy vs. no. of epochs')
```

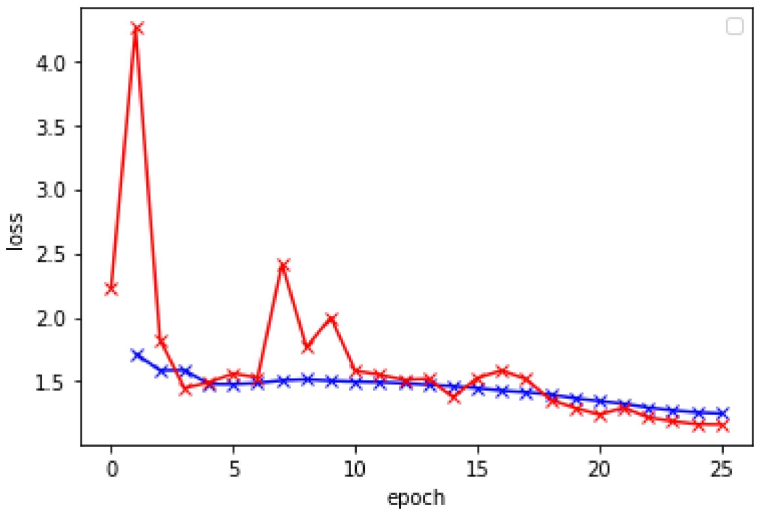
In [26]:
```
plot_acc(history)
```



In [27]:
```
def plot_losses(history):
    train_loss = [x.get('train_loss') for x in history]
    val_loss = [x['val_loss'] for x in history]
    plt.plot(train_loss, '-bx')
    plt.plot(val_loss, '-rx')
    plt.xlabel('epoch')
    plt.ylabel('loss')
    plt.legend('Training', 'Validation')
```

In [28]:
```
plot_losses(history)
```

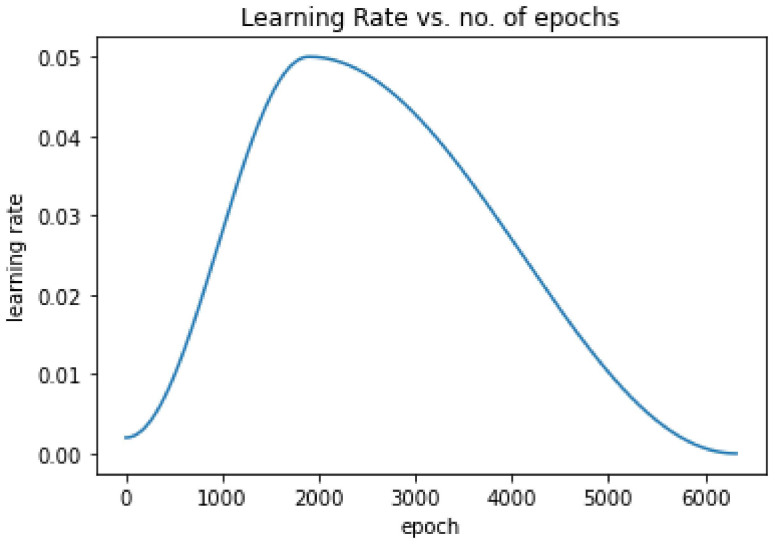
```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'T' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'r' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'a' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'i' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'n' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists

/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:8: UserWarning: Legend d
oes not support 'g' instances.
A proxy artist may be used instead.
See: http://matplotlib.org/users/legend_guide.html#creating-artists-specifically-for-
adding-to-the-legend-aka-proxy-artists
```



In [29]:

```python
def plot_lrs(history):
    lrs = np.concatenate([x.get('lrs',[]) for x in history])
    plt.plot(lrs)
    plt.xlabel('epoch')
    plt.ylabel('learning rate')
    plt.title('Learning Rate vs. no. of epochs')
```

In [30]:
```
plot_lrs(history)
```



In [ ]:
```

```